



Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e Computação – ICMC
Prof. Dr. Murilo Francisco Tomé

GABRIEL DE ANDRADE DEZAN – 10525706
IVAN MATEUS DE LIMA AZEVEDO – 10525602
RAFAEL GONGORA BARICCATTI – 10892273

1º Trabalho Prático – Métodos Iterativos para Sistemas Lineares

São Carlos – SP
2019

Introdução

Em Análise Numérica, os métodos iterativos são utilizados para resolver sistemas de equações lineares muito grandes. Um desses métodos é o método de Gauss-Seidel. Este trabalho consiste na solução de um sistema linear matricial da forma $A \cdot x = b$ através deste método.

O método de Gauss-Seidel é uma variação do método de Jacobi. Veremos a seguir o porquê desse método ter sido criado.

1. Método iterativo de Jacobi

O método de Jacobi funciona da seguinte forma: supondo um sistema linear nas incógnitas x_1, \dots, x_n da seguinte forma:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

Supondo também que todos os termos $a_{i,i}; i=1, \dots, n$ sejam diferentes de zero (se não for o caso, isso pode ser resolvido às vezes com uma troca na ordem de equações). Nesse caso, a solução desse sistema satisfaz:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}} [b_1 - a_{12}x_2 - \dots - a_{1n}x_n] \\ x_2 &= \frac{1}{a_{22}} [b_2 - a_{21}x_1 - \dots - a_{2n}x_n] \\ &\vdots \\ x_n &= \frac{1}{a_{nn}} [b_n - a_{n1}x_1 - \dots - a_{n,n-1}x_{n-1}] \end{aligned}$$

O método de Jacobi consiste em definir uma aproximação inicial $x^{(0)}$, onde $x = [x_1, \dots, x_n]^T$ é o vetor resposta do sistema linear na forma matricial $A \cdot x = b$, e substituí-la no lado direito das equações acima e obter assim o vetor $x^{(1)}$, em seguida substituir esses novos valores novamente para calcular o vetor $x^{(2)}$ e assim sucessivamente. Então tem-se:

$$\begin{aligned}x_1^{(k+1)} &= \frac{1}{a_{11}}[b_1 - a_{12}x_2^{(k)} - \dots - a_{1n}x_n^{(k)}] \\x_2^{(k+1)} &= \frac{1}{a_{22}}[b_2 - a_{21}x_1^{(k)} - \dots - a_{2n}x_n^{(k)}] \\&\vdots \\x_n^{(k+1)} &= \frac{1}{a_{nn}}[b_n - a_{n1}x_1^{(k)} - \dots - a_{n,n-1}x_{n-1}^{(k)}]\end{aligned}$$

No caso geral, para um elemento qualquer de $\mathbf{x}^{(k+1)}$, temos:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}x_j^{(k)}], \quad i=1, \dots, n, \quad k=0, 1, 2, \dots$$

1.1. Forma matricial do método iterativo de Jacobi

Sejam as matrizes D (matriz diagonal), L (matriz triangular inferior) e U (matriz triangular superior), tais que $A = L + D + U$. Multiplicando a equação (I) por a_{ii} e reestruturando a equação, podemos chegar na forma matricial do método:

$$\begin{aligned}x_i^{(k+1)} &= \frac{1}{a_{ii}}[b_i - \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}x_j^{(k)}] \\a_{ii}x_i^{(k+1)} &= b_i - \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}x_j^{(k)}\end{aligned}$$

Pode-se notar então que o equivalente matricial é:

$$\begin{aligned}(D)\mathbf{x}^{(k+1)} &= \mathbf{b} - (L+U)\mathbf{x}^{(k)} \\ \mathbf{x}^{(k+1)} &= -D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}\end{aligned}$$

Definem-se então, outras duas matrizes C_J e \mathbf{g}_J tais que $C_J = -D^{-1}(L+U)$ e $\mathbf{g}_J = D^{-1}\mathbf{b}$.

2. Método iterativo de Gauss-Seidel

O método de Gauss-Seidel então surge como uma forma de otimizar o método de Jacobi. A otimização surge a partir do seguinte raciocínio: supondo que o método de Jacobi seja convergente, podemos observar que para que possamos calcular

$\mathbf{x}^{(k+1)}$, precisamos dos valores de $\mathbf{x}^{(k)}$ e também que como a sequência $\mathbf{x}^{(k+1)}$ é convergente, então os valores $x_i^{(k+1)}, i=1, \dots, n$ que já foram calculados

são uma melhor aproximação que os valores $x_i^{(k)}$, $i=1,\dots,n$. Dessa forma temos então a definição do método iterativo de Gauss-Seidel:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right], \quad i=1,\dots,n, \quad k=0,1,2,\dots \quad (II)$$

Ou seja, para uma dada linha i a ser calculada, os valores de $x^{(k+1)}$ com índices menores que i já foram então previamente calculados e são melhores aproximações que os valores de $x^{(k)}$ de mesmo índice. Esse método constitui então um método melhor que o de Jacobi, pois ele converge mais rápido.

2.1. Forma matricial do método iterativo de Gauss-Seidel

Sejam as matrizes D (matriz diagonal), L (matriz triangular inferior) e U (matriz triangular superior), tais que $A=L+D+U$. Multiplicando a equação (II) por a_{ii} e reestruturando a equação, podemos chegar na forma matricial do método:

$$\begin{aligned} x_i^{(k+1)} &= \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right] \\ a_{ii} x_i^{(k+1)} &= b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \\ a_{ii} x_i^{(k+1)} + \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} &= b_i - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \\ \sum_{j=1}^i a_{ij} x_j^{(k+1)} &= b_i - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \end{aligned}$$

Pode-se notar então que o equivalente matricial é:

$$\begin{aligned} (L+D)x^{(k+1)} &= b - Ux^{(k)} \\ x^{(k+1)} &= -(L+D)^{-1} Ux^{(k)} + (L+D)^{-1} b \end{aligned}$$

Definem-se então, outras duas matrizes C_J e g_J tais que $C_{GS} = -(L+D)^{-1} U$ e $g_{GS} = (L+D)^{-1} b$.

3. Forma matricial geral

Sejam as matrizes M e N , tais que $A=M+N$, então temos que $C = -M^{-1}N$ e $g = M^{-1}b$, com $M_J = D$ e $N_J = L+U$ para o método de Jacobi e $M_{GS} = L+D$ e $N_{GS} = U$ para o método de Gauss-Seidel.

Em geral, para qualquer dos métodos iterativos, temos que:

$$\mathbf{x}^{(k+1)} = C \mathbf{x}^{(k)} + \mathbf{g} \quad (III)$$

4. Condição para convergência do método de Gauss-Seidel

Para saber sob quais condições o método converge, analisemos o comportamento o erro da sequência, definido por $\mathbf{e}^{(k+1)} = \mathbf{x} - \mathbf{x}^{(k+1)}$. Tomando as mesmas matrizes mostradas acima M e N , tais que $A = M + N$, temos:

$$\begin{aligned} A \mathbf{x} &= \mathbf{b} \\ (M + N) \mathbf{x} &= \mathbf{b} \\ M \mathbf{x} + N \mathbf{x} &= \mathbf{b} \\ \mathbf{x} &= -M^{-1} N \mathbf{x} + M^{-1} \mathbf{b} \end{aligned}$$

Subtraindo a equação obtida acima pela equação (III), temos:

$$\begin{aligned} \mathbf{x} - \mathbf{x}^{(k+1)} &= C \mathbf{x} - C \mathbf{x}^{(k+1)} \\ \mathbf{x} - \mathbf{x}^{(k+1)} &= C (\mathbf{x} - \mathbf{x}^{(k+1)}) \\ \mathbf{e}^{(k+1)} &= C \mathbf{e}^{(k)}, \forall k \end{aligned}$$

Então, como a equação acima vale para todo k , temos:

$$\begin{aligned} k=0 &\Rightarrow \mathbf{e}^{(1)} = C \mathbf{e}^{(0)} \\ k=1 &\Rightarrow \mathbf{e}^{(2)} = C \mathbf{e}^{(1)} = C \cdot C \mathbf{e}^{(0)} = C^2 \mathbf{e}^{(0)} \\ k=2 &\Rightarrow \mathbf{e}^{(3)} = C \mathbf{e}^{(2)} = C \cdot C^2 \mathbf{e}^{(0)} = C^3 \mathbf{e}^{(0)} \\ &\vdots \end{aligned}$$

Em geral, temos: $\mathbf{e}^{(k+1)} = C^{k+1} \mathbf{e}^{(0)}, \forall k$. Portanto, a única forma de o vetor erro convergir para o vetor nulo é se $\lim_{k \rightarrow \infty} C^{k+1} = 0$, ou seja, se a matriz C for uma matriz convergente.

Outras formas de se verificar se o método iterativo converge são também formas de provar que a matriz é convergente. Uma das formas é através da norma da matriz, outra, do raio espectral e outra, verificando se ela é diagonal estritamente dominante.

4.1. Outras formas: norma da matriz

Tirando a norma da equação que define o erro acima, temos:

$$\begin{aligned} \|\mathbf{e}^{(k+1)}\| &= \|C^{k+1} \mathbf{e}^{(0)}\| \\ \|\mathbf{e}^{(k+1)}\| &\leq \|C^{k+1}\| \cdot \|\mathbf{e}^{(0)}\| \end{aligned}$$

Mas podemos observar que:

$$\begin{aligned}\|C^{k+1}\| &= \|C \cdot C^k\| \leq \|C\| \cdot \|C^k\| \\ \|C^{k+1}\| &\leq \|C\| \cdot \|C^k\| \leq \|C\| \cdot \|C\| \cdot \|C^{k-1}\| \leq \|C\| \cdot \|C\| \cdot \|C^{k-1}\| \\ \|C^{k+1}\| &\leq \|C\|^2 \cdot \|C^{k-1}\|\end{aligned}$$

Em geral, temos então:

$$\|C^{k+1}\| \leq \|C\|^{k+1}$$

Logo, pelo teorema da comparação de sequências, se $\lim_{k \rightarrow \infty} \|C\|^{k+1} = 0$, a sequência $\|C^{k+1}\|$ converge. E a única forma de o limite acima ser verdade é se $\|C\| < 1$. Portanto, se isso for cumprido, o método converge.

4.2. Outras formas: raio espectral da matriz

O raio espectral de uma matriz é dado por:

$$\rho(A) = \max\{|\lambda_i|, i=1, \dots, n\}$$

Onde λ_i são os autovalores da matriz. Então temos o seguinte teorema: dada uma matriz $A_{n \times n}$, a matriz é convergente se, e somente se, $\rho(A) < 1$. Portanto, se $\rho(C) < 1$, C é uma matriz convergente e, portanto, o método converge.

4.3. Outras formas: diagonal estritamente dominante

Para que uma matriz seja dita diagonal estritamente dominante, deve-se

cumprir o seguinte: seja $A_{n \times n}$ uma matriz qualquer, se $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|$, então a

matriz é dita estritamente dominante por colunas e se $|a_{jj}| > \sum_{\substack{i=1 \\ i \neq j}}^n |a_{ij}|$, então a matriz

é dita estritamente dominante por linhas. Ela é dita estritamente dominante se cumprir uma dessas duas condições ou as duas. No caso dos métodos iterativos, se a matriz C for estritamente dominante, então o método converge.

Problema

A matriz de coeficientes dada no problema é uma matriz pentadiagonal definida por (I) :

$$(I) \begin{cases} a_{i,i}=5, i=1,\dots,n \\ a_{i,i+1}=-1, i=1,\dots,n-1 \\ a_{i+1,i}=-1, i=1,\dots,n-1 \\ a_{i,i+3}=-1, i=1,\dots,n-3 \\ a_{i+3,i}=-1, i=1,\dots,n-3 \\ a_{i,j}=0, \text{no restante} \end{cases}$$

Para $n=6$, por exemplo, temos a matriz:

$$A = \begin{bmatrix} 5 & -1 & 0 & -1 & 0 & 0 \\ -1 & 5 & -1 & 0 & -1 & 0 \\ 0 & -1 & 5 & -1 & 0 & -1 \\ -1 & 0 & -1 & 5 & -1 & 0 \\ 0 & -1 & 0 & -1 & 5 & -1 \\ 0 & 0 & -1 & 0 & -1 & 5 \end{bmatrix}$$

Pode-se mostrar que essa matriz é SPD. Considere o método iterativo de Gauss-Seidel:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right], \quad i=1,\dots,n, \quad k=0,1,2,\dots$$

- (a) Escreva um subprograma que, tendo como dados de entrada uma matriz real A , um vetor real b , um inteiro n , uma constante real ε e uma constante inteira $itmax$, utiliza o método de Gauss-Seidel e obtém aproximações $x^{(k+1)}$ da solução do sistema $Ax=b$, até que $\|x^{(k+1)} - x^{(k)}\|_{\infty} \leq \varepsilon$. A variável $itmax$ determina o número máximo de iterações permitido. Se $((k+1) > itmax)$, parar o programa e imprimir a mensagem: METODO GAUSS-SEIDEL DIVERGIU.

- (b) Para testar o programa, faça $n=50,100$ e $b_i = \sum_{j=1}^n a_{ij}$, $i=1,\dots,n$, $\varepsilon=10^{-8}$ e $itmax=5n$ e execute o programa. A solução obtida deve ser $x_i=1$, $i=1,\dots,n$.

(c) Utilizando o subprograma desenvolvido no item (a), resolva o sistema $A\mathbf{x}=\mathbf{b}$, onde A é uma matriz definida pelas equações (I) e \mathbf{b} é o vetor definido por $b_i=\frac{1.0}{i}$, $i=1,\dots,n$. Considere o caso $n=500$ e $\varepsilon=10^{-8}$. Partindo da aproximação inicial $\mathbf{x}^{(0)}=\mathbf{0}$ obtenha a solução do sistema linear pelo método de Gauss-Seidel. Deve-se também escrever uma função para calcular a norma infinita de um dado vetor e usá-lo para calcular $\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\|_{\infty}\leq\varepsilon$.

Implementação

A linguagem utilizada para escrever o programa foi a linguagem C. Visto que a matriz de coeficientes dada pelo problema é uma matriz esparsa, ou seja, uma matriz com uma grande quantidade de valores nulos comparados aos não-nulos, foi optado por utilizar uma implementação de matriz esparsa que guarda somente os valores não-nulos.

Essa escolha se deu pelo fato de que, como os valores nulos não causariam diferença nos cálculos, não há motivo para se desperdiçar memória guardando-os. Portanto, foi utilizada uma estrutura de dados especial para guardar a matriz. A estrutura pode ser observada logo abaixo:

```
//----Defining the struct of a node----
typedef struct node{ //Structure of a node
    int i;
    int j;
    long double data;
    struct node *nextRow;
    struct node *nextCol;
} node;

//----Defining the struct of a row and a column----
typedef struct node* list;

//----Defining the struct of a matrix----
typedef struct sparse_matrix{ //Structure of a matrix
    list *rows;
    list *cols;
} matrix;
```

Figura 1: Definição da estrutura de dados utilizada.

Então a matriz se constitui de duas listas encadeadas de nós, de forma que a lista chamada *rows* guarda os ponteiros das linhas e a *cols*, das colunas. Além disso, cada lista dessa é composta por vários nós que guardam os índices do elementos não-nulos da matriz, além do seu valor em si e dos ponteiros pros próximos elementos da linha e da coluna. Um esquema de uma matriz definida pelas equações (I) com $n=4$:

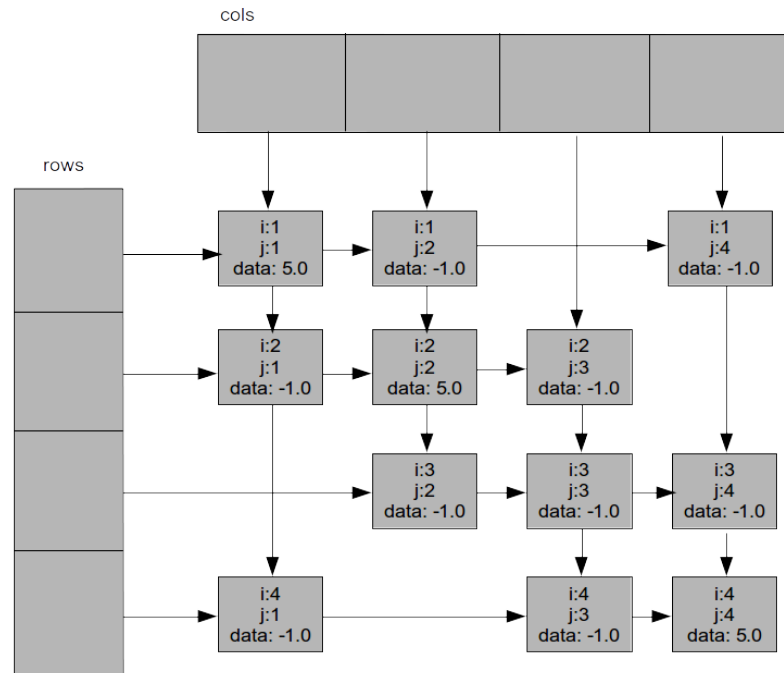


Figura 2: Esquema da estrutura de dados utilizada.

Logo abaixo serão explicadas as funções que foram utilizadas para implementar o método e que estão no código. São elas:

- `node *init_node(int i, int j, long double data)`: esta função recebe como parâmetros os índices de um dado elemento da matriz e o seu valor e inicializa um nó a ser inserido na matriz esparsa.
- `list *init_list(int n)`: esta função recebe o parâmetro `n` que informa tamanho da lista que será utilizada e inicializa essa lista.
- `matrix *init_matrix(int n, int m)`: esta função recebe como parâmetros as dimensões de uma dada matriz e a inicializa.
- `void insert_row(list *head, node *new)`: esta função recebe como parâmetros o primeiro elemento de uma lista (no caso a lista é uma linha) e um elemento a ser inserido e então o insere.
- `void insert_col(list *head, node *new)`: esta função faz a mesma operação que a anterior. A diferença é que a inserção é feita em uma coluna.
- `void insert_matrix(matrix *A, node *new, int i, int j)`: esta função recebe uma matriz, um nó a ser inserido e os índices desse nó e o insere na matriz.

- `matrix *initial_array(int n)`: esta função inicializa um vetor nulo novo (que nada mais é que uma matriz coluna) de n linhas.
- `long double test_matrix_A_values(int i, int j)`: esta função é a função que simula o sistema de equações dado no problema para definir quais os valores não-nulos da matriz, ou seja, ela recebe os índices desse vetor e determina se o valor do elemento é 5, -1 ou 0.
- `matrix *create_test_matrix_A(int n)`: esta função cria a matriz de coeficientes $n \times n$ dada no exercício para testar o código implementado.
- `long double test_matrix_b_values(matrix *A, int i, int test)`: esta função recebe três parâmetros: a matriz de coeficientes A , o índice de uma linha e uma flag chamada `test`. Primeiro a função verifica qual o valor da flag; se for 1, significa que o teste a ser realizado é o teste da letra (b) do trabalho, em que

$$b_i = \sum_{j=1}^n a_{ij}, \quad i=1, \dots, n. \quad \text{Se não for 1, então o teste é o da letra (c), em que}$$

$$b_i = \frac{1.0}{i}, \quad i=1, \dots, n. \quad \text{Então, dependendo da flag test, a função retorna um}$$

desses valores para um elemento b_i .

- `matrix *create_test_matrix(matrix *A, int n, int test)`: esta função utiliza a função anterior para criar o vetor de teste \mathbf{b} .
- `matrix *subtract_arrays(matrix *x1, matrix *x2, int n)`: esta função retorna um vetor que é a subtração de dois vetores $x1$ e $x2$.
- `long double abs_value(long double n)`: esta função simplesmente calcula o valor absoluto de uma variável do tipo `long double`.
- `long double inf_norm(matrix *x, int n)`: esta função calcula a norma infinita de um dado vetor x .
- `void copy_array(matrix *x1, matrix *x2, int n)`: esta função copia os valores do vetor $x2$ em $x1$ (utilizada para fazer a simulação de $\mathbf{x}^{(k)}$ e $\mathbf{x}^{(k+1)}$).
- `matrix *gauss_seidel(matrix *A, matrix *b, long double epsilon, int itmax, int n)`: esta função implementa o que foi pedido na letra (a) deste trabalho.
- `void destroy_matrix(matrix *A, int n)`: esta função recebe uma matriz de n de linhas e apaga essa matriz da memória.

Sabendo qual foi a estrutura e as funções utilizadas, temos o código do método implementado:

```
matrix *gauss_seidel(matrix *A, matrix *b, long double epsilon, int itmax, int n){
    int k = 0;
    long double bi = 0;
    long double aij = 0;
    long double x_kplusj = 0;
    long double x_kj = 0;
    long double sum1 = 0;
    long double sum2 = 0;
    long double aii = 0;
    node *aux = NULL;
    matrix *x_k = initial_array(n);
    matrix *x_kplus = initial_array(n);

    do{
        for(int i = 0; i < n; ++i){
            copy_array(x_k, x_kplus, n);
            sum1 = 0;
            sum2 = 0;
            bi = (b->rows[i])>data;
            aux = A->rows[i];
            while(aux != NULL && aux->j < i){
                aij = aux->data;
                x_kplusj = (x_kplus->rows[aux->j])>data;
                sum1 += aij * x_kplusj;
                aux = aux->nextCol;
            }
            aux = aux->nextCol;
            while(aux != NULL && aux->j < n){
                aij = aux->data;
                x_kj = (x_k->rows[aux->j])>data;
                sum2 += aij * x_kj;
                aux = aux->nextCol;
            }
            aux = A->rows[i];
            while(aux->j != i && aux != NULL){
                aux = aux->nextCol;
            }
            aii = aux->data;
            (x_kplus->rows[i])>data = (bi - sum1 - sum2)/aii;
        }
        ++k;
    }while(inf_norm(subtract_arrays(x_kplus,x_k,n), n) > epsilon && (k + 1) <= itmax);
    if(k + 1 > itmax && inf_norm(subtract_arrays(x_kplus,x_k,n), n) > epsilon) {
        printf("METODO GAUSS-SEIDEL DIVERGIU\n");
    }
    return x_kplus;
}
```

Figura 3: Implementação do método de Gauss-Seidel.

Resultados

A função *main* do programa consiste de um menu básico em que o usuário pode escolher que teste deseja realizar (o teste para $n=50$, $n=100$ e $n=500$). Após escolher a opção, o programa executará conforme instruído e serão impressos, em cada linha do terminal, os valores dos índices do vetor resultado aproximado $x^{(k+1)}$, além dos valores calculados.

Para o primeiro teste ($n=50$ e $b_i = \sum_{j=1}^n a_{ij}$, $i=1,\dots,n$), temos a seguinte imagem (que precisou ser cortada no elemento da linha 38):

```
TEST 1a
i: 1 | j: 1 | data: 0.9999995933
i: 2 | j: 1 | data: 0.9999994539
i: 3 | j: 1 | data: 0.9999993588
i: 4 | j: 1 | data: 0.9999992086
i: 5 | j: 1 | data: 0.9999991141
i: 6 | j: 1 | data: 0.9999990445
i: 7 | j: 1 | data: 0.9999989772
i: 8 | j: 1 | data: 0.9999989289
i: 9 | j: 1 | data: 0.9999988949
i: 10 | j: 1 | data: 0.9999988695
i: 11 | j: 1 | data: 0.9999988546
i: 12 | j: 1 | data: 0.9999988489
i: 13 | j: 1 | data: 0.9999988505
i: 14 | j: 1 | data: 0.9999988593
i: 15 | j: 1 | data: 0.9999988744
i: 16 | j: 1 | data: 0.9999988950
i: 17 | j: 1 | data: 0.9999989206
i: 18 | j: 1 | data: 0.9999989505
i: 19 | j: 1 | data: 0.9999989842
i: 20 | j: 1 | data: 0.9999990213
i: 21 | j: 1 | data: 0.9999990611
i: 22 | j: 1 | data: 0.9999991032
i: 23 | j: 1 | data: 0.9999991471
i: 24 | j: 1 | data: 0.9999991924
i: 25 | j: 1 | data: 0.9999992387
i: 26 | j: 1 | data: 0.9999992855
i: 27 | j: 1 | data: 0.9999993325
i: 28 | j: 1 | data: 0.9999993792
i: 29 | j: 1 | data: 0.9999994254
i: 30 | j: 1 | data: 0.9999994707
i: 31 | j: 1 | data: 0.9999995150
i: 32 | j: 1 | data: 0.9999995578
i: 33 | j: 1 | data: 0.9999995990
i: 34 | j: 1 | data: 0.9999996384
i: 35 | j: 1 | data: 0.9999996760
i: 36 | j: 1 | data: 0.9999997114
i: 37 | j: 1 | data: 0.9999997447
i: 38 | j: 1 | data: 0.9999997759
```

Figura 4: Vetor resultado para o primeiro teste.

Para o segundo teste ($n=100$ e $b_i = \sum_{j=1}^n a_{ij}$, $i=1,\dots,n$), temos a seguinte

imagem (que precisou ser cortada no elemento da linha 38):

TEST 1b		
i: 1	j: 1	data: 0.9999995736
i: 2	j: 1	data: 0.9999994245
i: 3	j: 1	data: 0.9999993201
i: 4	j: 1	data: 0.9999991562
i: 5	j: 1	data: 0.9999990481
i: 6	j: 1	data: 0.9999989638
i: 7	j: 1	data: 0.9999988793
i: 8	j: 1	data: 0.9999988121
i: 9	j: 1	data: 0.9999987571
i: 10	j: 1	data: 0.9999987083
i: 11	j: 1	data: 0.9999986677
i: 12	j: 1	data: 0.9999986338
i: 13	j: 1	data: 0.9999986048
i: 14	j: 1	data: 0.9999985805
i: 15	j: 1	data: 0.9999985600
i: 16	j: 1	data: 0.9999985428
i: 17	j: 1	data: 0.9999985283
i: 18	j: 1	data: 0.9999985161
i: 19	j: 1	data: 0.9999985059
i: 20	j: 1	data: 0.9999984974
i: 21	j: 1	data: 0.9999984902
i: 22	j: 1	data: 0.9999984842
i: 23	j: 1	data: 0.9999984792
i: 24	j: 1	data: 0.9999984751
i: 25	j: 1	data: 0.9999984716
i: 26	j: 1	data: 0.9999984687
i: 27	j: 1	data: 0.9999984663
i: 28	j: 1	data: 0.9999984643
i: 29	j: 1	data: 0.9999984626
i: 30	j: 1	data: 0.9999984613
i: 31	j: 1	data: 0.9999984601
i: 32	j: 1	data: 0.9999984592
i: 33	j: 1	data: 0.9999984584
i: 34	j: 1	data: 0.9999984579
i: 35	j: 1	data: 0.9999984574
i: 36	j: 1	data: 0.9999984571
i: 37	j: 1	data: 0.9999984568
i: 38	j: 1	data: 0.9999984568

Figura 5: Vetor resultado para o segundo teste.

Para o terceiro teste ($n=500$ e $b_i = \frac{1.0}{i}$, $i=1,\dots,n$), temos a seguinte

imagem (que precisou ser cortada no elemento da linha 38):

```

TEST 2
i: 1 | j: 1 | data: 0.2898702619
i: 2 | j: 1 | data: 0.2341134459
i: 3 | j: 1 | data: 0.1896667126
i: 4 | j: 1 | data: 0.2152470780
i: 5 | j: 1 | data: 0.1910406083
i: 6 | j: 1 | data: 0.1656513770
i: 7 | j: 1 | data: 0.1556703488
i: 8 | j: 1 | data: 0.1402042227
i: 9 | j: 1 | data: 0.1252260484
i: 10 | j: 1 | data: 0.1144056475
i: 11 | j: 1 | data: 0.1040978964
i: 12 | j: 1 | data: 0.0947716352
i: 13 | j: 1 | data: 0.0870475643
i: 14 | j: 1 | data: 0.0802122995
i: 15 | j: 1 | data: 0.0741664884
i: 16 | j: 1 | data: 0.0689380175
i: 17 | j: 1 | data: 0.0643335032
i: 18 | j: 1 | data: 0.0602560027
i: 19 | j: 1 | data: 0.0566543542
i: 20 | j: 1 | data: 0.0534491281
i: 21 | j: 1 | data: 0.0505812139
i: 22 | j: 1 | data: 0.0480077909
i: 23 | j: 1 | data: 0.0456869701
i: 24 | j: 1 | data: 0.0435841649
i: 25 | j: 1 | data: 0.0416716118
i: 26 | j: 1 | data: 0.0399249355
i: 27 | j: 1 | data: 0.0383234943
i: 28 | j: 1 | data: 0.0368500166
i: 29 | j: 1 | data: 0.0354896417
i: 30 | j: 1 | data: 0.0342296398
i: 31 | j: 1 | data: 0.0330591283
i: 32 | j: 1 | data: 0.0319687059
i: 33 | j: 1 | data: 0.0309502308
i: 34 | j: 1 | data: 0.0299966361
i: 35 | j: 1 | data: 0.0291017541
i: 36 | j: 1 | data: 0.0282601805
i: 37 | j: 1 | data: 0.0274671648
i: 38 | j: 1 | data: 0.0267185132

```

Figura 6: Vetor resultado para o terceiro teste.

Por fim, como esperado, o método converge muito rápido e não precisa de muitas iterações para tanto. Como sabemos o resultado dos dois primeiros testes ($x_i=1.0$, $i=1,\dots,n$), podemos afirmar que, para esses dois, o resultado obtido foi realmente bastante acurado. Já no terceiro teste, mesmo não tendo o resultado como nos dois primeiros, confiamos que a precisão está também muito boa, visto que a implementação está correta. Assim, os resultados também o estão.